2025-04-16 self hosting dreamwidth with docker

objectively insane title i know

yeah we're doing this. or in other words, i already did this 6 months ago. so by we i mean i'm helping YOU learn how to self host dreamwidth with docker because you can do it bare metal but it didn't work for me when i tried so i just got crazier[1] and did docker compose instead.

let's get this out of the way: dreamwidth is forked from livejournal and built in perl and a custom markup language and the latter happened because a better one didn't exist yet. this stuff's all older than me. the dreamwidth developers do fantastic work stripping out the old BML — Better Markup Language, he really called it that — and replacing it with semi-readable[2] efficient perl code. i want to start all of this by mentioning how kind and helpful and considerate the DW devs all are, how they entertained *and* voluntarily helped me with my stupid shitpost idea of self hosting DW on a dell optiplex IN DOCKER. step by step, every time i got stuck on a part of perl or DW or anything that they could help with, they helped. and i promised this documentation to them as soon as i finished my initial deploy of the site, after 25 tossed mySQL databases and tons of failed edited dockerfiles, and it's come so late that i'm embarrassed. i hope this is sufficient enough.

baseline stuff

this guide will assume you know docker quite well. if you don't you should really familiarize yourself because we're not doing the light shit like pulling from docker hub and then running pre-made stuff. no we're building slightly edited dockerfiles and running slapdash scripts into containers like fucking animals. this is a terribly disorganized experience and if you don't know the docker basics it will be absolutely miserable.

this guide will NOT assume that you know perl because frankly i don't either. perl scares the shit out of me. i struggle with the constraints and abstractions of fucking ruby on rails. you think i can work with perl? all i know is that the dollar sign is a variable and the at sign is an array and a friend told me that. this language is terrifying and ungodly. i want to learn it.

this guide will assume that you know basic git. i did not know basic git when i did this, minus git clone, and i only recently started version controlling my DW deploy[1-1]. git knowledge will help you a lot here. please version control your site.

and lastly, this guide will assume you know the extreme basics of SQL. by that i mean you know `SHOW TABLES;` and `SELECT * FROM sitekeywords;` and shit. we're not doing some SQL rocket science here and i'll mostly guide you through stuff but if i don't explain those parts it's because i barely know what the hell it's doing and i copied it from stack overflow and it worked.

important resources

1. dreamwidth wiki: oh my god please save PDFs of the pages of this that i link to or something because you will want copies of this. absolutely invaluable stuff. the production notes page offhandedly mentioned interacting with `ddlockd` over telnet and it changed my life. save that and the scratch installation page at the very least.
2. dreamwidth github repo: make a github account if you don't have one and search that repo to your heart's content. if you need to debug something you at least have the power of God and the 13 year old code comments in perl or BML files on your side.
3. my sourcehut repo for stuff i did that's referenced in this guide: read ahead to learn about the files in this!

table of contents

getting started

the compose file

alright so here is by far the most embarrassing part of all of this: sharing my compose file. everyone point and laugh at this absolute nonsense:

```
services:
  web:
    container_name: web
    build:
      context: /home/dw/dw/etc/docker/web
      dockerfile: Dockerfile
    ports:
      - "3237:80"
    volumes:
      - /home/dw/dw/etc/docker/web/files/config-
private.pl:/dw/etc/config-private.pl
      - /home/dw/dw/etc/docker/web/files/config-
local.pl:/dw/etc/config-local.pl
      - /home/dw/dw/etc/docker/web/files/config.pl:/dw/etc/config.pl
      - /home/dw/dw/etc/texttool.pl:/dw/etc/texttool.pl
      - /home/dw/dw/etc/build-static.sh:/dw/etc/build-static.sh
      - /home/dw/dw/cgi-bin/DW/TaskQueue.pm:/dw/cgi-
bin/DW/TaskQueue.pm
      - /home/dw/dw/var/taskqueue:/dw/var/taskqueue:rw
      - /home/dw/dw/cgi-bin/DW/TaskQueue/LocalDisk.pm:/dw/cgi-
bin/DW/TaskQueue/LocalDisk.pm
      - /home/dw/dw/blobimages:/dw/var/blobimages:rw
      - /home/dw/dw/bin/worker-manager:/dw/bin/worker-manager
      -
/home/dw/dw/etc/docker/worker/files/workers.conf:/dw/etc/workers.conf
      -
/home/dw/dw/htdocs/stc/gradation/gradation.css:/dw/htdocs/stc/gradatio
n/gradation.css:rw
      - /home/dw/dw/htdocs/scss/skins/gradation/_gradation-
base.scss:/dw/htdocs/scss/skins/gradation/_gradation-base.scss:rw
      - /home/dw/dw/cgi-bin/DW/SiteScheme.pm:/dw/cgi-
bin/DW/SiteScheme.pm:rw
      - /home/dw/dw/cgi-bin/DW/SiteScheme.pm:/dw/ext/dw-nonfree/cgi-
```

```
bin/DW/Hooks/SiteScheme.pm:rw
      -
/home/dw/dw/htdocs/img/profile_icons:/dw/htdocs/img/profile_icons
      - /home/dw/dw/bin/upgrading/en.dat:/dw/bin/upgrading/en.dat:rw
      - /home/dw/dw/bin/upgrading/base-
data.sql:/dw/bin/upgrading/base-data.sql:rw
      -
/home/dw/dw/bin/upgrading/proplists.dat:/dw/bin/upgrading/proplists.da
t:rw
      - /home/dw/dw/cgi-bin/DW/Controller/Index.pm:/dw/ext/dw-
nonfree/cgi-bin/DW/Controller/Dreamwidth/Index.pm
      - /home/dw/dw/views/index-free.tt:/dw/views/index-free.tt
    depends_on:
       mysql:
         condition: service_healthy

  worker:
    container_name: worker
    build:
      context: /home/dw/dw/etc/docker/worker
      dockerfile: Dockerfile
    volumes:
      - /home/dw/dw/etc/docker/web/files/config-
private.pl:/dw/etc/config-private.pl
      - /home/dw/dw/etc/docker/web/files/config-
local.pl:/dw/etc/config-local.pl
      - /home/dw/dw/etc/docker/web/files/config.pl:/dw/etc/config.pl
      - /home/dw/dw/bin/worker-manager:/dw/bin/worker-manager
      -
/home/dw/dw/etc/docker/web/files/workers.conf:/dw/etc/workers.conf
      - /home/dw/dw/cgi-bin/DW/TaskQueue.pm:/dw/cgi-
bin/DW/TaskQueue.pm
      - /home/dw/dw/cgi-bin/DW/TaskQueue/LocalDisk.pm:/dw/cgi-
bin/DW/TaskQueue/LocalDisk.pm
      - /home/dw/dw/var/taskqueue:/dw/var/taskqueue:rw
    command: bash -c "/dw/bin/worker-manager --debug"
    depends_on:
      mysql:
        condition: service_healthy

  lock:
    container_name: lock
    build:
      context: /home/dw/dw/etc/docker/worker
      dockerfile: Dockerfile
    environment:
      - PERL5LIB=/dw/extlib/lib/perl5
    command: bash -c "/dw/bin/ddlockd"
    ports:
      - "7002:7002"
```

```
  mysql:
    container_name: db
    build:
      context: /home/dw/dw/etc/docker/mysql-build
      dockerfile: Dockerfile
    env_file: .env
    command: --sql_mode=""
    volumes:
      - ./mysql25:/var/lib/mysql
      - /home/dw/dw/etc/docker/cnf/my.cnf:/etc/my.cnf:ro
    ports:
      - "3306:3306"
    healthcheck:
      test: ["CMD-SHELL", "ls" ]
      start_period: 10s
      interval: 5s
      timeout: 5s
      retries: 3
```

PLEASE believe me when i say this used to look far worse. there used to be an entirely superfluous migrations container here that did absolutely nothing but make restart times unnecessarily long and i have removed all reference of it and instead added the `lock` container which is critical for basic functions.

the reason why i have all of those files and folders mounted is because i could not for the life of me get each container to mount the entire cloned root folder it was in. it just would not cooperate. if you want to edit a file and persist it, you have to mount it manually into the container. it is a massive pain in the ass and this would look way more normal if i could do it the normal way. but i digress.

by the way, if you haven't git cloned the dreamwidth repo by now, first, make a dedicated user on your server for dreamwidth. i called mine `dw`. this will make life easier for reasons i forgot. then, clone the repo into `/var/www` and not the home folder unless you're a sicko like me. from here, change directories to the `etc/docker` folder because that's where we're gonna live.

when editing this compose file to reflect your file structure, you can probably just find and replace `/home/dw/dw/` to either `/var/www/root_name` if not in home, or if you are, `/home/USER/root_name`; in both of these, change `root_name` to the name of the root folder that you cloned (by default it will be `dreamwidth`), and in the case of the home directory, change `USER` to your dedicated user's name.

also, please edit `~/.profile` for the new user and add this line in:

```
export LJHOME=/home/USER/root_name # or /var/www/root_name
```

be sure to change `USER` to your user's name. LJ/DW code relies on this variable and it's included in every dockerfile but it's probably important to have it set bare metal too.

so we're going to be building every docker image here. even mySQL but that one barely counts because the dockerfile for that one is just pulling from a mySQL image anyway and adding a bit more to it. so based on that, let's start simple with the database.

dockerfiles

db

```
FROM mysql:5.7
ENV MYSQL_DATABASE dw
COPY . /docker-entrypoint-initdb.d
```

if you're thinking "you can't be serious" you are right. all we're doing is copying an entry point in here and calling an environment variable! that's it! i don't remember why i did either of those things but follow along.

we have to use mySQL version 5.7 because i tried mySQL version 8.0 and it kept dying on me and the dreamwidth developers said that the code supports up to 5.7 currently.

anyway the contents of the entry point folder are now completely unnecessary but please build the image anyway for the sake of that ENV variable. it will probably be important later.

you may notice the `.env` file called for this container. this is critical because you need to set a database password here. PLEASE FOR THE LOVE OF GOD PROTECT THIS FILE. `chmod` that shit and never ever push it to git or your site is fucked until you change things.

here's my redacted version of the file:

```
MYSQL_DATABASE=USER
MYSQL_USER=USER
MYSQL_PASSWORD=PASS
MYSQL_ROOT_PASSWORD=PASS
```

```
MYSQL_PORT=3306
MYSQL_HOST=db
```

change `USER` to your DB username of choice, and `PASS` to a hardened password that you will save because it is a pain in the ass to change it. you'll need this password after we're done with the dockerfiles so put it in a password manager or something.

we're going to keep going in order of simplicity, so we're moving onto the `lock` container.

lock

`lock` uses an existing dockerfile from the dreamwidth github repo, as the rest of these do, but is modified and custom built for reasons i forgot (you're going to keep hearing this, it's been a while). it's the `worker` image, because the `base` image didn't provide enough for me to launch the lockfile server from, and `web` was overkill. so this one it is.

i had my instance running for about 6-7 months with a weird, stubborn little error i couldn't debug: every time you would edit and save your profile, the page would hang. if you navigated to another page, and checked back on your profile, you'd notice that everything but the list of interests saved. this was a non-issue to me until i realized that this also impacted the registration flow, as you edit and save your profile there as well. users would get stuck on the page hang, and the confirmation email could never get sent. i had to manually force password reset emails in their place. it was miserable and no one liked it.

april 2025 comes. i finally dedicate time to debugging this with a wonderful dreamwidth volunteer. the root of it? FUCKING LOCK FILES FROM BEFORE I WAS BORN.

`ddlockd` is a tiny little perl binary that runs a telnet-interactive server and creates lockfiles so that data doesn't collide into each other and corrupt. it's older than me by a year or two and was written by Brad himself and some other guy at one of livejournal's companies. it's the load bearing mac mini[3] of this shit.

so all you need for this image is the contents of the `worker` image, because that will provide you with enough perl and DW stuff that you can put `command: bash -c "bin/ddlockd"` in the compose file (it's already there! because i did this!) and the humble little binary will do its lockfile job.

worker

speak of the devil! alright so if you've done website stuff you're probably aware of what workers and jobs are. they're little tasks that get scheduled and done automatically for you. these tasks keep your site going. for example, the email worker will... send your email. every worker labeled ESN will do notifications. and so on. these are important.

i don't need to paste the contents of this, as stated in the previous section, because it's on github. i didn't need to modify this one. just go there.

web

`web` is the web server and more. it's the important front-end and where i mounted most files including configuration and shit. my version of the dockerfile contains so much commented code i can't share it without looking stupid so i will instead refer you to the github copy because that one is actually sensible and everything i did in the dockerfile wound up needing to be done externally.

database

in my notes for this i have the following lines:

> `docker compose up -d`
> it crashes
> `docker compose up -d mysql`

this is quite literally what you will be doing. just copy the compose file up there, build the images with `docker compose build`, pray to god that there's no errors, and hopefully you will get to this point. this section is some funky SQL shit. be ready.

first off, we have to make a mySQL configuration file. this will set the bind address to 0.0.0.0 and the protocol as TCP, making it easier to connect to it outside of docker. the configuration can be anywhere, but has to be mounted to `/etc/my.cnf`. be sure that it's read only with the `:ro` suffix on the mount:

```
[mysqld]
disable-skip-name-resolve
host_cache_size=0
bind-address=0.0.0.0

[client]
protocol=tcp
```

anyway your database container should be up by now. you have to get into a bash shell for it, as follows:

```
docker compose exec -it mysql bash
```

once you're in there, go find the root DB password you saved earlier when doing the mySQL env file stuff, and copy it. then, run this command, and when it prompts you for the password, paste it:

```
mysql -h localhost -P 3306 --protocol=tcp -u root -p
```

after that, you SHOULD be into a mySQL shell now! yay! unfortunately we have to do some password stuff! aw!

this has to be done because mySQL will later change how passwords are done, and we have to make sure this database retains the old way of doing it, because if things are ever upgraded, we don't want to get locked out.

i'm going to paste some mySQL commands that you have to run sequentially, and the only thing you have to change are the password values, which are identified in the commands by `MYSQL_ROOT_PASSWORD_CHANGEME` and `MYSQL_PASSWORD_CHANGEME` and they are asking to be changed so give them what they want please. if it's not obvious, these values have to match `MYSQL_ROOT_PASSWORD` and `MYSQL_PASSWORD` from the env file respectively.

```
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY
'MYSQL_ROOT_PASSWORD_CHANGEME';

ALTER USER 'root'@'%' IDENTIFIED WITH mysql_native_password BY
'MYSQL_ROOT_PASSWORD_CHANGEME';

CREATE USER 'dw'@'localhost' IDENTIFIED WITH mysql_native_password BY
'MYSQL_PASSWORD_CHANGEME';

ALTER USER 'dw'@'%' IDENTIFIED WITH mysql_native_password BY
'MYSQL_PASSWORD_CHANGEME';
```

all we're doing is the future proofing for the password stuff, and then creating a new user to interact with the database here.

after this, run a simple `flush privileges;` to kind of clear the slate.

NOW we can make databases!

run these each:

```
create database dw;

GRANT ALL PRIVILEGES ON dw.* to 'dw'@'localhost';

GRANT ALL PRIVILEGES ON dw.* to 'dw'@'%';

create database dw_schwartz;

GRANT ALL PRIVILEGES ON dw_schwartz.* to 'dw'@'localhost';

GRANT ALL PRIVILEGES ON dw_schwartz.* to 'dw'@'%';
```

`dw` is your main database, while `dw_schwartz` is for workers i think. refer to scratch installation for more details.

for the schwartz DB, you have to populate it with schema data. exit out of your mySQL shell back into the bash shell and do that as follows:

```
mysql -h localhost -P 3306 --protocol=tcp -uroot -p dw_schwartz < $LJHOME/doc/schwartz-schema.sql
```

it will prompt you for the `root` password, so give it that, and then it will do its thing.

you'd think we should populate the main DB now, but for that, we have to take a brief detour into config files.

config files

so for this i am going to first link the scratch installation section on it and instruct you to follow what it says. do all of it. if there's anything that needs to be changed you'll come back to it.

ok i'm assuming you've done all of that now. moving on here's my changes, step by step:

config.pl

`$DOMAIN_EMAIL = "gmail.com";` : set this domain to whatever your sender email is. i'm using a mailjet SMTP relay with a gmail address for this site because i've set up a mail server once and i am not doing that shit ever again God bless. but if you are rolling your own email or using something different, put it here.

`# email addresses [...]` : you SHOULD in theory change every email variable here except the bogus one (that should stay at that `null` one), but for me i just changed `ADMIN` and `ABUSE` to the first part of my sender gmail address. you don't need to add the domain in.

`#$DEFAULT_CAPTCHA_TYPE = "T";` : un-comment this, we need a captcha for stuff. no it's not recaptcha or any of that dumb traffic light shit. DW chose the normal sane route for a fallback and went with textcaptcha, which is super easily configured. we'll get to that later.

and last, at the bottom of the file, we need a custom line:

```
if ($LJ::IS_DEV_SERVER == 0) {
  %PASSWORD_PEPPER_KEYS = ( 2 => pack('H*', 'CHANGEME', ));
  $PASSWORD_PEPPER_KEY_CURRENT_ID = 2;

}
```

BIG thanks to the DW devs for helping me here because this is crazy important and i had no idea it was. basically, when running a production instance, passwords need to be secured or hashed or something. the pepper thing, right. this does a weird math thing with a value you will have to generate and then replace `CHANGEME` with it. generate that value with the below command in your shell:

```
perl -e 'print unpack("H*", join( "", map { chr(rand(256)) } 1..32 ))'
```

thanks to alierak in the DW discord for supplying this to me! they also advised that the key be rotated every once in a while for security, so jot that down.

take the value that perl command generates and put it where `CHANGEME` is in that above statement

config-local.pl

`$IS_DEV_SERVER = 0;` : this will be set to 1 by default. we're doing production so set it at 0.

```
$SITENAME = "love4eva";
$SITENAMESHORT = "love4eva";
$SITENAMEABBREV = "l4eva";
$SITECOMPANY = "lol";
```

all of the above can be changed to what the variables ask for. i kind of just winged it but these are used in the views.

right under where it asks for cache servers, put this in:

```
@LOCK_SERVERS = ('lock:7002');
```

this is how the web and worker containers will be able to connect with the lock container. INCREDIBLY important!

```
$SMTP_SERVER = "in-v3.mailjet.com";
$MAIL_TO_THESCHWARTZ = 1;
```

set these both to your SMTP server addresses — NOT the address domain! it has to be to the server! mailjet has its own relay server so i used that domain here, not my sender address' domain of course. and set the variable after to 1 so it's enabled.

config-private

here's the big one. we're configuring the database connection now.

yours is going to look like this; make sure the rest of the `%DBINFO` stuff is commented out except this:

```
%DBINFO = (
           'master' => {  # master must be named 'master'
               'host' => 'db',
               'port' => 3306,
               'user' => 'CHANGEMEUSER',    # CHANGETHIS if on
Dreamhack to dh_username
               'pass' => 'CHANGEMEPASS',    # CHANGETHIS
               'dbname' => 'CHANGEMEDBNAME',    # CHANGETHIS if on
Dreamhack to dreamhack_username
               'role' => {
                   'cluster1' => 1,
                   'slow' => 1,
```

```
                    # optionally, apache write its access logs to a
mysql database
                    #logs => 1,
                },
            },
```

change the user, pass, and DB name variables to the corresponding values for your database. you did this above so just find where you saved them.

scroll down a bit in the file, and you'll find the configuration for the schwartz worker DB:

```
@THESCHWARTZ_DBS = (
        {
            dsn => 'dbi:mysql:dw_schwartz;host=db', # CHANGETHIS if on
Dreamhack to dreamhack_username instead of dw_schwartz
            user => 'CHANGEMEUSER_WORKER', # CHANGETHIS if on
Dreamhack to dh_username
            pass => 'CHANGEMEPASS_WORKER',     # CHANGETHIS
            port => 3306,
        },
    );
```

again, change the user and pass values here to the corresponding values, but not the same ones as above; these are for your schwartz DB. the DSN value can stay but note that the `host=db` part has to match the container name, so that the web container can connect to it. docker networking is weird.

back to email stuff!

```
%EMAIL_VIA_SES = (
    hostname => 'in-v3.mailjet.com',
    username => 'API_KEY_OR_USER',
    password => 'SECRET_OR_PASSWORD',
);
```

these are only necessary if you're using a relay like me. hostname is the same as the SMTP server from above, while username and password will be the API key and secret that your address is provided by your relay. these are also basically the username and password respectively.

ok, so here's a fork in the road: are you going to use amazon S3 for your images, or use a local storage implementation for your images that is quite literally called "inefficient" by the person who wrote it at the top of the file[4]? i can't help you if you choose S3 because i chose blob store.

the local disk module is written to "just work" and it does work, yes, but barely. every icon and image on site will load horribly slow. it's not efficient at all and i chose it because i don't know S3 but i'm honestly going to advise that you learn it because it will make your site usable. but if you for whatever reason want to do it on disk, here's how i did it.

first, un-comment the local disk part of `@BLOBSTORES` and be sure to comment the rest out (except for the closing parenthesis of course). you'll be left with this:

```
 @BLOBSTORES = (
     # Local disk configuration, can be used to store everything on
one machine
     localdisk => {
         path => "/dw/var/blobimages",
     },
```

`/dw` is an absolute path here, but will be mounted in the compose file to wherever you want. for me i put it in the root of the cloned folder as `blobimages`, but in the container it will be the path you set it as in this config file. for the record, everything in the container will live in `/dw`.

next you'll want to disable hCaptcha, because screw that shit:

```
$CAPTCHA_HCAPTCHA_SITEKEY = undef;
$CAPTCHA_HCAPTCHA_SECRET = undef;
```

make sure these are both `undef` to disable it.

now we're at the private part of the config! oh boy i sure hope none of this is important! (it is)

`$DOMAIN` is here. put this as your website domain. please note that you CANNOT use a subdomain for your site. i had to dedicate a whole domain to this site because DW relies on subdomains for journals. so put your domain as name dot TLD. for example, love4eva.net.

the rest can remain commented, except for `%TEXTCAPTCHA` at the very end. this is how we configure the sane captcha. it's very easy thankfully:

```
%TEXTCAPTCHA = (
#   # this works for testing purposes.
#   # sign up at the textcaptcha website for a key for production use
    api_key => "LITERALLY_ANY_EMAIL",
    timeout => 10,
);
```

yeah so as the comments say, go to the site for this service, sign up with, as the key variable says, LITERALLY ANY EMAIL OF YOURS, and they will give it a key. you can then just paste the same email for the API key and it SHOULD be functional. at least it is for me.

more database stuff but it's not too bad

so let's refer back to the scratch installation page and do what it says, from a shell in the web container (`docker compose exec -it web bash`, if you forgot):

```
$LJHOME/bin/upgrading/update-db.pl -r --innodb
$LJHOME/bin/upgrading/update-db.pl -r --innodb # at least for now we
have to run this twice
$LJHOME/bin/upgrading/update-db.pl -r --cluster=all --innodb
$LJHOME/bin/upgrading/update-db.pl -p
```

run those sequentially. it will do all the stuff for the main DB. the page says this is when you populate the worker DB but my notes say do that earlier so whatever.

as the page says, we gotta make a system account. it says you can change this later but i forgot if that's actually true so uh please just make and save a password and then paste it when the script asks for it:

`$LJHOME/bin/upgrading/make_system.pl`

lastly we gotta run a tool that'll load translations and words and stuff. this is important ok. you'll see that this has to be run every time you restart your site later but i'll get to why when the time comes:

`$LJHOME/bin/upgrading/texttool.pl load`

FIRST RUN OF THE SITE YEAHHHHHHH

WE'RE FINALLY FUCKING HERE. OK. RUN THIS:

```
docker compose up -d --force-recreate
```

and ALL of your containers should start, and your database will be restarted too. this will take a while. usually for me it takes a minute or two. let it do its thing.

now don't get too excited, we have to get a shell into the web container and do stuff to get style sheets in order:

```
docker compose exec -it web bash
```

```
gem install compass --version 1.0.3 && gem install sass --version 3.4.25
```

this does some SCSS installation that's critical for your site.

now, exit out of the shell, and run `up -d` again, and go to your domain and HOLY SHIT! THERE'S NOTHING THERE!

never mind we have to reverse proxy now

ok so if you use nginx or traefik or something i can't help you because i am a CADDY WARRIOR. i'm assuming you know how to use those as reverse proxies if you do use them but in case you don't, here's some guidance for caddy.

let's get this clear: as i mentioned, the DW code relies on subdomains for every journal. in theory, on paper, you should have on demand TLS set up for this using cloudflare or something. in practice.... i just manually load every single subdomain on the same line and let caddy create a zeroSSL certificate for all of them:

```
example1.love4eva.net, example2.love4eva.net, example3.love4eva.net {
    tls YOUR_EMAIL_HERE {
        ca https://acme.zerossl.com/v2/DV90
    }

    reverse_proxy IP:3237 {
        header_up X-Real-Ip {remote_host}
    }
```

```
    log {
        output file /var/log/caddy/access.log {
            roll_size 1gb
            roll_keep 5
            roll_keep_for 720h
        }
    }
}
```

yes this is fucking insane[1-2]. my reasoning for not doing on demand TLS like a normal person? well for starters, caddy is kind of annoying about setting this up. you have to install your domain provider's API module and build the caddy binary with it. also, your domain provider has to have good API access and allow on demand TLS. guess who uses a domain provider that doesn't do either of those things: ya girl!

i did look into moving my domain to cloudflare, but at the time of me doing all of this, my domain was too new to be moved and i had to wait another month. so i told myself this would be a temporary thing.... and then it wasn't.

honestly like just use zeroSSL and keep your site small if you're doing this. i have at most 15-20 people on my site and my real config for this looks gnarly. don't be me, who ignored the suggestions[5], PLEASE use on demand TLS and keep yourself sane if you're building a big site.

also, real quick, you're going to have to do a redirect for the wildcard to www. this means no www to www and is as simple as this in caddy:

```
https://love4eva.net {
    redir https://www.love4eva.net
}
```

configuring workers

for your emails to send, and other things, you need to configure the mounted `workers.conf` file. note that `worker-manager` is also mounted; this doesn't need to be edited but it's necessary that it's mounted for it to be ran in the worker container.

`workers.conf` is pretty modular and well-described with comments, so if you follow the comments, you can really customize it, but on its own, you just have to add the workers

`send-email-ses` and `dw-send-email` to the "all" section at the top of the configuration file.

all that said, here's what my `workers.conf` file looks like:

```
all:
    esn-fired-event: 1
    esn-process-sub: 1
    send-email: 1
    esn-cluster-subs: 1
    lazy-cleanup: 1
    import-scheduler: 1
    content-importer: 1
    send-email-ses: 1
    dw-send-email: 1
```

if using local disk modules

you may have noticed a few strange files mounted in the compose file, such as `dw/cgi-bin/DW/TaskQueue/LocalDisk.pm` and `dw/cgi-bin/DW/TaskQueue.pm`. if you did go with local disk storage for your images, these mounts are necessary and `TaskQueue.pm` file will require minor edits to make icon and image uploading smooth.

i honestly can't remember WHY i did these, but i tested on my site and removing them does indeed mess things up for image uploading so just make the edits i don't know.

in `TaskQueue.pm`, comment the following lines in the `sub get` block near the top:

```
    $_queue = DW::TaskQueue::LocalDisk->init();

    # Determine what kind of queue object to build, depending on if
we're
    # running locally or not
    if ( exists $LJ::SQS{region} ) {
        return $_queue = DW::TaskQueue::SQS->init(%LJ::SQS);
    }
```

i really cannot remember why this needed to be commented but again, shit broke on my site when i had it not commented. fuck around and find out if you want i guess.

then, in the following line that checks if the server is in development mode or not, change `if ($LJ::IS_DEV_SERVER)` to force it to work in production:

```
if ($LJ::IS_DEV_SERVER == 0) {
    return $_queue = DW::TaskQueue::LocalDisk->init();
}
```

this one i have mostly forgotten, but i can assume that, since local disk was meant to work only for development, this and the prior tweak force it to work in production.

customization of the site

the header makes this sound like it's optional but it's really not so i'm gonna bold text the following:

do NOT use the following dreamwidth themes on your site, and especially don't customize them! these are NOT free for usage and dreamwidth doesn't intend for them to be used by independent sites!

- Tropospherical Red
- Tropospherical Purple

the non-free code, as it's often referred to on the dreamwidth wiki and even still within the code, used to be in its own github repository, but apparently the two were merged at some point. regardless, **there ARE themes in the code that you can use**! just not the ones above. the list of ones you can use and customize are below:

- Celerity
- Gradation Horizontal
- Gradation Vertical
- Lynx (this one is text only)

these are both separated in the code anyway — you'll only find the CSS and SCSS for the non-free Tropospherical themes in the `$LJHOME/ext/dw-nonfree/htdocs` directory — but i wanted to make this clear.

as i've alluded to, the themes aren't built with raw CSS, of course. SCSS is compiled, in theory, with every restart of the web container. in practice, this does not happen and i have a workaround, but that will come later.

now: customizing these themes is a major pain if you do not know SCSS! it's very confusing and i had to hop between the CSS and SCSS files to make things align. the split is also present because of the BML conversion that's in progress for dreamwidth; the BML pages cannot use the SCSS i don't even think it existed yet so it's plain CSS for them. that's why some dreamwidth pages look slightly different and more or less modern than others.

personally, i chose to customize Gradation Horizontal for my site, flipping it to a light theme and making it pink because i love pink. for your site, you can pick from these existing themes and customize them to your heart's content, as i did of course, but if you want something REAL messy to start you off with, i've shared my custom versions of Gradation Horizontal in a sourcehut repo. this repo is linked in the resources section for quick reference.

default site scheme

to set the default site theme, or the site scheme, as it's called, we have to do something kinda hacky. you might have noticed these weird mounts in the compose file:

```
- /home/dw/dw/cgi-bin/DW/SiteScheme.pm:/dw/cgi-bin/DW/SiteScheme.pm:rw
- /home/dw/dw/cgi-bin/DW/SiteScheme.pm:/dw/ext/dw-nonfree/cgi-
bin/DW/Hooks/SiteScheme.pm:rw
```

notice how the same file is mounted twice, but the second time, it's mounted over a different file? that's the hacky thing.

what happened here was that `cgi-bin/DW/SiteScheme.pm` had been edited to reflect the default scheme. however, as it stands, non-free code is what controls the *default AND logged out site scheme*. thus, if a logged out user views the site without this mount, they would see non-free dreamwidth branding! and we can't do that!

so i tried this hack during initial deploy and it somehow worked. do not ask me how but it did. anyway here's what the edited part of my version of that file looks like, to make Gradation Horizontal the only available theme:

```
package DW::SiteScheme;
use strict;

my %sitescheme_data = (
 #   blueshift             => { parent => 'common', title    =>
"Blueshift" },
#    celerity              => { parent => 'common', title    =>
```

```
"Celerity" },
    common                 => { parent => 'global', internal => 1 },
    'gradation-horizontal' => { parent => 'common', title    =>
"Gradation Horizontal" },
    'gradation-vertical'   => { parent => 'common', title    =>
"Gradation Vertical" },
    lynx                   => { parent => 'common', title    => "Lynx
(light mode)" },
    global                 => { engine => 'current' },
    tt_runner              => { engine => 'bml',    internal => 1 },
);
```

notice the comments. you can make blueshift and celerity options if you want of course, i just decided not to. but this file already didn't include the Tropo themes — the non-free DW themes — and i guess my weird hail mary mount made things work for me so whatever i hope it does for you too.

custom index controller

in the middle of writing this guide i went to clear up some things with DW devs about the non-free code i mentioned above, and i realized i had some sitting in my compose file! it was a file called `index.tt.text.local`, and after a quick look at it on my server and then on the DW github repo, i realized it was there to edit some dreamwidth hardcoding. turns out, that hardcoding was being rendered by MORE non-free code!

so i talked it out and we realized that the non-free code would take precedence over the free versions because the non-free part has its own index page controller. that controller makes it assume priority. momijizukamori, the wonderful DW volunteer who's helped me lots with this whole project, wrote a basic controller that loads the free version of that index page, but static.

since it was static, it would show the same page to both logged in and logged out users. so if you wanted logged in users to see their inbox and notifications, they wouldn't get that, and logged out users would be presented with a basic set of links meant for logged in users. so i got my hands dirty!

i wrote perl code for the first time for the sake of these docs. it was terrifying. perl is an ungodly language. i respect it though for its power. honestly half of the code is redundant and most of it is copied and learned from other dreamwidth modules. but i tried my best, and it does the job! this controller code and the new index page are both in the sourcehut repo, but here's the controller anyway, because it's so short:

`$LJHOME/cgi-bin/DW/Controller/Index.pm`:

```
#!/usr/bin/perl
#
# DW::Controller::Index
#
# Controller for the site homepage.

package DW::Controller::Index;

use strict;
use warnings;

use DW::Routing;
use DW::Template;
use DW::Controller;
use DW::Panel;
use DW::Widget::QuickUpdate;

DW::Routing->register_string( '/index', \&indexfree_handler,
app => 1 );

sub indexfree_handler {
    my ( $ok, $rv ) = controller( anonymous => 1 );
    return $rv unless $ok;

    my $remote = $rv->{remote};
    my $stuff;
    my $widget;

    if ($remote) {
       $stuff->{remote} = $remote;
        $stuff->{panel} = DW::Panel->init( u => $remote );

        $widget->{primary} = DW::Widget::QuickUpdate->render_body;
        $stuff->{helpme} = DW::Panel->_render( $widget );
    }

    else {
        $stuff->{panel} = DW::Panel->init( u => $remote );
    }

    return DW::Template->render_template( 'index-free.tt', $stuff );
}

#DW::Routing->register_static( '/index',           'index-free.tt',
app => 1 );

1;
```

custom text

somehow i figured out how to add custom social media properties for users to put in their profiles, so here's a quick note on the several files you'll need to edit to get those.

the files are as follows in the compose file:

```
- /home/dw/dw/bin/upgrading/en.dat:/dw/bin/upgrading/en.dat:rw
- /home/dw/dw/bin/upgrading/base-data.sql:/dw/bin/upgrading/base-
data.sql:rw
-
/home/dw/dw/bin/upgrading/proplists.dat:/dw/bin/upgrading/proplists.da
t:rw
- /home/dw/dw/htdocs/img/profile_icons:/dw/htdocs/img/profile_icons
```

let's say we're going to add a custom field for bluesky (i did this already). starting with `en.dat`, you have to add a line as follows:

```
profile.service.bluesky=Bluesky
```

and put it where the other lines that start with `profile.service` are. i have mine in somewhat alphabetical order.

`base-data.sql` looks pretty gnarly, but it's not too bad. you just have to sift through the lines and choose whether you want to add a free text field or a link field. in the bluesky case i opted for free text, as bluesky usernames are dynamic and can feature domain names so a direct link can't necessarily work as easily. so i'll show examples for both bluesky (free text) and dreamwidth (link):

```
INSERT IGNORE INTO profile_services (name, userprop, imgfile,
title_ml, url_format, maxlen) VALUES ('bluesky', 'bluesky',
'bluesky.png', 'profile.service.bluesky', NULL, 40);
UPDATE profile_services SET userprop='bluesky', imgfile='bluesky.png',
title_ml='profile.service.bluesky', url_format=NULL, maxlen=40 WHERE
name='bluesky';
```

notice how `url_format` is NULL up above, and the presence of `maxlen`, which defines the max characters. we'll get to the images in the next one.

for dreamwidth, which has a link format, it goes a tiny bit differently:

```
INSERT IGNORE INTO profile_services (name, userprop, imgfile,
title_ml, url_format, maxlen) VALUES ('dreamwidth', 'dreamwidth',
'dreamwidth.png', 'profile.service.dreamwidth', '//%s.dreamwidth.org',
30);
UPDATE profile_services SET userprop='dreamwidth',
imgfile='dreamwidth.png', title_ml='profile.service.dreamwidth',
url_format='//%s.dreamwidth.org', maxlen=30 WHERE name='dreamwidth';
```

this time `url_format` is defined! i simply mimicked some existing entries for other sites and realized that `%s` can stand for the subdomain that dreamwidth requires, and the protocol is not required.

up next is `proplists.dat`. this one looks more complex but all you have to do is follow a template like this:

```
userproplist.bluesky:
cldversion: 4
datatype: char
des: Bluesky handle
indexed: 0
multihomed: 0
prettyname: Bluesky
```

that's about it. oh and for the little images that go next to these entries on the profile, they're mounted in this compose file on the host at `$LJHOME/htdocs/img/profile_icons`. the images must be 16x16 pixels. i usually just take the favicon of the site and shrink it further if needed.

running your site

WE'RE FINALLY HERE! YOU CAN RUN YOUR SITE!

so every time you restart the containers, your style sheet will reset to the defaults before your edits. i don't know why it does this but it's annoying. also, if you made any additions to the profile social media lists, those disappear. i've made a script that i execute after the containers are started up, directed into a bash shell. this script will also handily generate site statistics for you, so it's kind of hitting two birds with one stone:

```
#!/bin/bash

# updates phrases (has to run twice) then builds stylesheets
```

```
$LJHOME/bin/upgrading/texttool.pl load &&
$LJHOME/bin/upgrading/texttool.pl load && $LJHOME/bin/build-static.sh

# updates stats
(http://wiki.dwscoalition.org/wiki/index.php/Statistics_setup)

$LJHOME/bin/ljmaint.pl genstats # Generate nightly stats
$LJHOME/bin/ljmaint.pl genstats_size # Generate site size stats
$LJHOME/bin/ljmaint.pl genstats_weekly # Generate weekly stats
$LJHOME/bin/ljmaint.pl genstatspics # Generate stat graphs
```

i have the script saved in `$LJHOME/etc/docker` as `up.sh`, so name it something like that, give it execute permissions, and run it as follows:

```
docker compose exec -T web bash < up.sh
```

admin stuff on site

before we wrap up, i'm going to cover something called privileges. there's a whole wiki page dedicated to this, but in short: the system account from earlier? that's critical for doing a lot of things, and it needs and can grant privileges (think of them as permissions) to users. log in as the system user, go to the admin console (`/admin/console`), and run the following one after another:

```
priv grant payments system

priv grant finduser system

priv grant suspend system

priv grant reset_email system

priv grant reset_password system
```

i don't remember why some of these are important but just go with it ok. if you find that your system account is missing some privileges, or you want to give your main user account some for ease of use (i've done this for making mood themes hahah), just refer to the wiki to see what you can grant.

CONCLUSION

oh god this is long and i am tired. i'm not saying anything for myself i will simply thank EVERY SINGLE DREAMWIDTH DEVELOPER FOR HELPING ME AND BEING KIND. ALSO ALL OF MY FRIENDS FOR BEING SUPPORTIVE IN THIS NONSENSE. i love you all i hope someone gets something out of this stupid guide.

references

girljpg

I will succeed because I'm crazy. 2025 mantra

1. ↩ ↩ ↩
2. from perl dot com itself ↩
3. do not unplug the load bearing mac mini! ↩
4. "grossly inefficient that just (kinda sorta) works" ↩
5. 

↩